**List Comprehensions**

**Why:** Produce a condensed, concise version of a “for” loop. Map an expression over a sequence to return a list of results.

**Syntax:** [*expression* for *target* in *iterable*]

**Example:** [*x \*\* 2* for *x* in *range(10)*]

**“For” loop equivalent:**

for *x* in *range(10)*:

*x \*\* 2*

**List comprehensions can include conditions.**

**Example:**

[*x \*\* 2* for *x* in *range(10)* if *x % 2 == 0*]

**“For” loop equivalent:**

for x in range(10):

if x % 2 == 0:

x \*\* 2

**Generators**

**Why:** To create in iterable function or expression to lighten the load on memory by producing one result at a time.

Generator *functions* are similar to regular functions but instead use “yield” instead of “return”.

**Yield:** Produces one result at a time instead of all at once

Generator *expressions* are syntactically identical to list comprehensions, but they are wrapped in parentheses ( ) instead of brackets [ ].

Both generators are iterables, meaning they are objects that can be sequenced through, much like a string or a list.

Please look at my jupyter notebook, as this isn’t enough space to demonstrate what I mean!